



National Textile University

Department of Computer Science

Subject:

Operating System

Submitted to:

Sir Nasir Mehmood

Submitted by:

Eman Babar

Reg. number:

23-NTU-CS-FL-1148

Semester:

5th- A

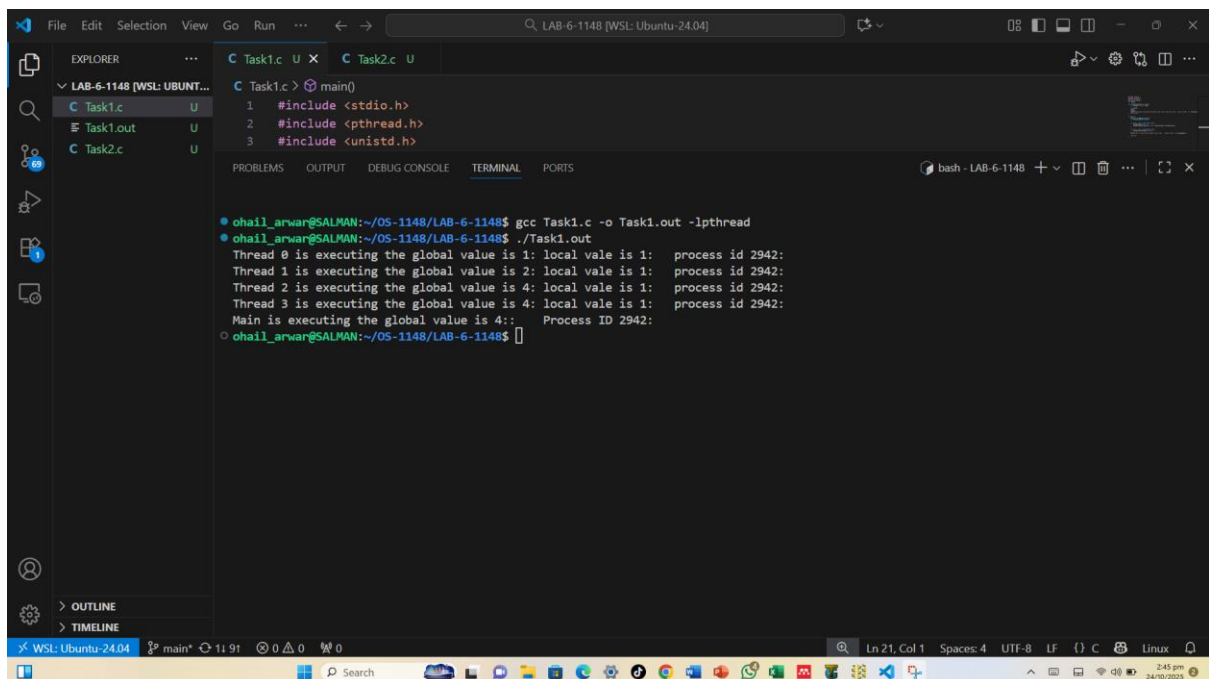
LAB-6

TASK-1

Code:

```
1 #include <stdio.h>
2 #include <pthread.h>
3 #include <unistd.h>
4 #define NUM_THREADS 4
5 int var=0;
6
7 void *thread_function(void *arg) {
8     int thread_id = *(int *)arg;
9
10    int varl=0;
11    var++;
12    varl++;
13    printf("Thread %d is executing the global value is %d: local vale is %d:   process id %d: \n", thread_id,varg,varl,getpid());
14    return NULL;
15 }
16
17 int main() {
18     pthread_t threads[NUM_THREADS];
19     int thread_args[NUM_THREADS];
20
21
22     for (int i = 0; i < NUM_THREADS; ++i) {
23         thread_args[i] = i;
24         pthread_create(&threads[i], NULL, thread_function, &thread_args[i]);
25     }
26
27     for (int i = 0; i < NUM_THREADS; ++i) {
28         pthread_join(threads[i], NULL);
29     }
30     printf("Main is executing the global value is %d::   Process ID %d: \n",varg,getpid());
31
32     return 0;
33 }
```

Output:



```
File Edit Selection View Go Run ... LAB-6-1148 [WSL: Ubuntu-24.04]
EXPLORER Task1.c Task2.c
LAB-6-1148 [WSL: UBUNT... Task1.c Task2.c
Task1.c Task1.out Task2.c
C Task1.c > main()
1 #include <stdio.h>
2 #include <pthread.h>
3 #include <unistd.h>
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
bash - LAB-6-1148
oohail_arvarg@SALMAN:~/OS-1148/LAB-6-1148$ gcc Task1.c -o Task1.out -lpthread
oohail_arvarg@SALMAN:~/OS-1148/LAB-6-1148$ ./Task1.out
Thread 0 is executing the global value is 1: local vale is 1:   process id 2942:
Thread 1 is executing the global value is 2: local vale is 1:   process id 2942:
Thread 2 is executing the global value is 4: local vale is 1:   process id 2942:
Thread 3 is executing the global value is 4: local vale is 1:   process id 2942:
Main is executing the global value is 4::   Process ID 2942:
oohail_arvarg@SALMAN:~/OS-1148/LAB-6-1148$
```

TASK-2-Critical Section

Code:

```
1  #include <stdio.h>
2  #include <pthread.h>
3  #include <unistd.h>
4  #define NUM_ITERATIONS 1000000
5  int count=10;
6  // Critical section function
7  void critical_section(int process) {
8      //printf("Process %d is in the critical section\n", process);
9      //sleep(1); // Simulate some work in the critical section
10     if(process==0){
11         for (int i = 0; i < NUM_ITERATIONS; i++)
12             count--;
13     }
14     else
15     {
16         for (int i = 0; i < NUM_ITERATIONS; i++)
17             count++;
18     }
19 }
20 void *process0(void *arg) {
21     // Critical section
22     critical_section(0);
23     // Exit section
24     return NULL;
25 }
26 void *process1(void *arg) {
27     // Critical section
28     critical_section(1);
29     // Exit section
30     return NULL;
31 }
32 int main() {
33     pthread_t thread0, thread1, thread2, thread3;
34     // Create threads
35     pthread_create(&thread0, NULL, process0, NULL);
36     pthread_create(&thread1, NULL, process1, NULL);
37     pthread_create(&thread2, NULL, process0, NULL);
38     pthread_create(&thread3, NULL, process1, NULL);
39     // Wait for threads to finish
40     pthread_join(thread0, NULL);
41     pthread_join(thread1, NULL);
42     pthread_join(thread2, NULL);
43     pthread_join(thread3, NULL);
44     printf("Final count: %d\n", count);
45     return 0;
46 }
```

Output:

The screenshot shows a Visual Studio Code editor window titled "LAB-6-1148 [WSL: Ubuntu-24.04]". The Explorer sidebar on the left shows a file tree with "Task.out", "Task1.c", "Task1.out", "Task2.c", and "Task2.out". The main editor area displays the code for "Task2.c", which includes a critical_section function and a main function. The terminal at the bottom shows the following commands and output:

```
ohail_arwar@SALMAN:~/OS-1148/LAB-6-1148$ gcc Task2.c -o Task2.out -lpthread
ohail_arwar@SALMAN:~/OS-1148/LAB-6-1148$ ./Task2.out
Final count: -323943
ohail_arwar@SALMAN:~/OS-1148/LAB-6-1148$ gcc Task2.c -o Task2.out -lpthread
ohail_arwar@SALMAN:~/OS-1148/LAB-6-1148$ ./Task2.out
Final count: -299518
ohail_arwar@SALMAN:~/OS-1148/LAB-6-1148$ gcc Task2.c -o Task2.out -lpthread
ohail_arwar@SALMAN:~/OS-1148/LAB-6-1148$ ./Task2.out
Final count: 283264
ohail_arwar@SALMAN:~/OS-1148/LAB-6-1148$
```

TASK-3-Peterson

Code:

```

1  #include <stdio.h>
2  #include <pthread.h>
3  #include <unistd.h>
4  #define NUM_ITERATIONS 100000
5  // Shared variables
6  int turn;
7  int flag[2];
8  int count=0;
9  // Critical section function
10 void critical_section(int process) {
11     //printf("Process %d is in the critical section\n", process);
12     //sleep(1); // Simulate some work in the critical section
13     if(process==0){
14         for (int i = 0; i < NUM_ITERATIONS; i++)
15             count--;
16     }
17     else
18     {
19         for (int i = 0; i < NUM_ITERATIONS; i++)
20             count++;
21     }
22     // printf("Process %d has updated count to %d\n", process, count);
23     //printf("Process %d is leaving the critical section\n", process);
24 }
25 // Peterson's Algorithm function for process 0
26 void *process0(void *arg) {
27     flag[0] = 1;
28     turn = 1;
29     while (flag[1]==1 && turn == 1) {
30         // Busy wait
31     }
32     // Critical section
33     critical_section(0);
34     // Exit section
35     flag[0] = 0;
36     //sleep(1);
37     pthread_exit(NULL);
38 }
39 // Peterson's Algorithm function for process 1
40 void *process1(void *arg) {
41     flag[1] = 1;
42     turn = 0;
43     while (flag[0] ==1 && turn == 0) {
44         // Busy wait
45     }
46     // Critical section
47     critical_section(1);
48     // Exit section
49     flag[1] = 0;
50     //sleep(1);
51     pthread_exit(NULL);
52 }
53 int main() {
54     pthread_t thread0, thread1;
55     // Initialize shared variables
56     flag[0] = 0;
57     flag[1] = 0;
58     turn = 0;
59     // Create threads
60     pthread_create(&thread0, NULL, process0, NULL);
61     pthread_create(&thread1, NULL, process1, NULL);
62     // Wait for threads to finish
63     pthread_join(thread0, NULL);
64     pthread_join(thread1, NULL);
65     printf("Final count: %d\n", count);
66     return 0;
67 }

```

Output:

```
File Edit Selection View Go Run ... LAB-6-1148 [WSL: Ubuntu-24.04]
EXPLORER
LAB-6-1148 [WSL: UBUNT...
  Task.out U
  Task1.c U
  Task1.out U
  Task2.c U
  Task2.out U
  Task3.c
  Task3.out
C Task3.c > ...
1 #include <stdio.h>
2 #include <pthread.h>
3 #include <unistd.h>
4 #define NUM_ITERATIONS 100000
5 // Shared variables
6 int turn;
7 int flag[2];
8 int count=0;
9 // Critical section function

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
ohail_arwar@SALMAN:~/OS-1148/LAB-6-1148$ gcc Task3.c -o Task3.out
ohail_arwar@SALMAN:~/OS-1148/LAB-6-1148$ ./Task3.out
Final count: 0
ohail_arwar@SALMAN:~/OS-1148/LAB-6-1148$

WSL: Ubuntu-24.04 main 11:91 0 0 0 0 Ln 2, Col 13 Spaces: 4 UTF-8 LF {} C Linux
```

TASK-4-Mutex

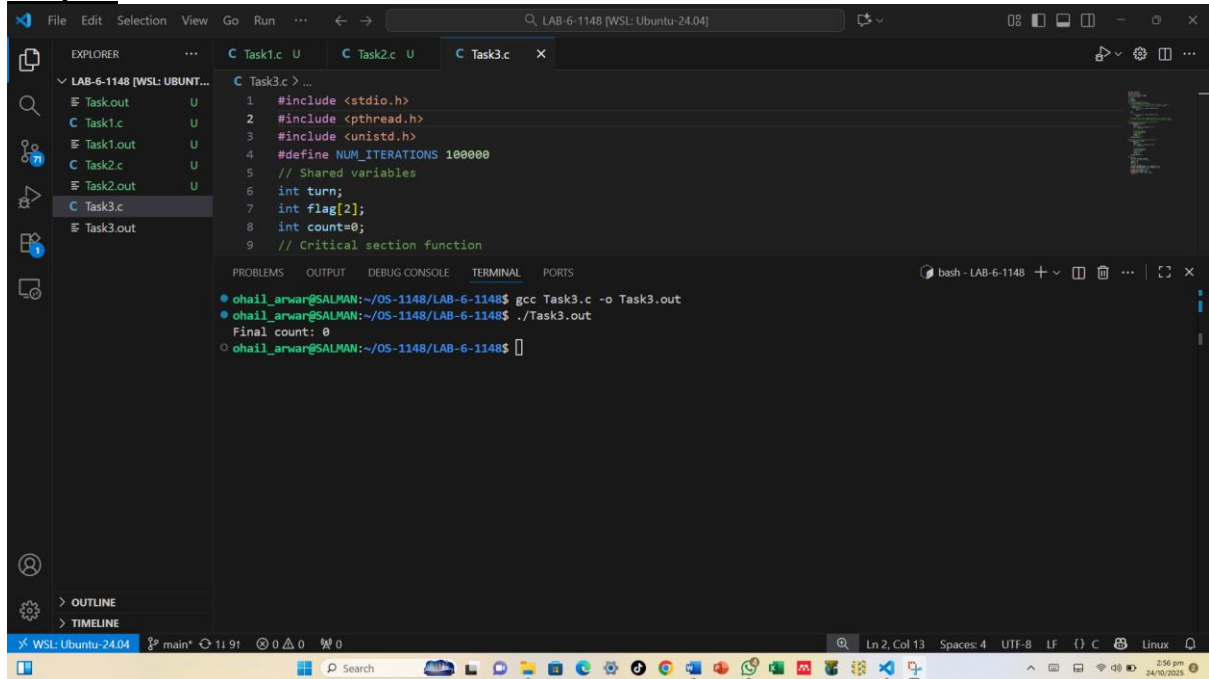
Code:

```

1  #include <stdio.h>
2  #include <pthread.h>
3  #include <unistd.h>
4  #define NUM_ITERATIONS 1000000
5  int count=10;
6  pthread_mutex_t mutex; // mutex object
7  // Critical section function
8  void critical_section(int process) {
9      //printf("Process %d is in the critical section\n", process);
10     //sleep(1); // Simulate some work in the critical section
11     if(process==0){
12
13         for (int i = 0; i < NUM_ITERATIONS; i++)
14             count--;
15     }
16     else
17     {
18         for (int i = 0; i < NUM_ITERATIONS; i++)
19             count++;
20     }
21     //printf("Process %d has updated count to %d\n", process, count);
22     //printf("Process %d is leaving the critical section\n", process);
23 }
24 // Peterson's Algorithm function for process 0
25 void *process0(void *arg) {
26     pthread_mutex_lock(&mutex); // lock
27     // Critical section
28     critical_section(0);
29     // Exit section
30     pthread_mutex_unlock(&mutex); // unlock
31     return NULL;
32 }
33 // Peterson's Algorithm function for process 1
34 void *process1(void *arg) {
35     pthread_mutex_lock(&mutex); // lock
36     // Critical section
37     critical_section(1);
38     // Exit section
39     pthread_mutex_unlock(&mutex); // unlock
40     return NULL;
41 }
42 int main() {
43     pthread_t thread0, thread1, thread2, thread3;
44     pthread_mutex_init(&mutex, NULL); // initialize mutex
45     // Create threads
46     pthread_create(&thread0, NULL, process0, NULL);
47     pthread_create(&thread1, NULL, process1, NULL);
48     pthread_create(&thread2, NULL, process0, NULL);
49     pthread_create(&thread3, NULL, process1, NULL);
50     // Wait for threads to finish
51     pthread_join(thread0, NULL);
52     pthread_join(thread1, NULL);
53     pthread_join(thread2, NULL);
54     pthread_join(thread3, NULL);
55     pthread_mutex_destroy(&mutex); // destroy mutex
56     printf("Final count: %d\n", count);
57     return 0;
58 }

```

Output:



The screenshot shows the Visual Studio Code interface with a C program named `Task3.c` open. The program includes `stdio.h`, `pthread.h`, and `unistd.h`, and defines `NUM_ITERATIONS` as 100000. It contains a critical section function. The terminal output shows the compilation and execution of the program, resulting in a final count of 0.

```
1 #include <stdio.h>
2 #include <pthread.h>
3 #include <unistd.h>
4 #define NUM_ITERATIONS 100000
5 // Shared variables
6 int turn;
7 int flag[2];
8 int count=0;
9 // Critical section function
```

```
ohail_arwar@SALMAN:~/05-1148/LAB-6-1148$ gcc Task3.c -o Task3.out
ohail_arwar@SALMAN:~/05-1148/LAB-6-1148$ ./Task3.out
Final count: 0
ohail_arwar@SALMAN:~/05-1148/LAB-6-1148$
```

TASK-5

Code:


```

1  #include <stdio.h>
2  #include <pthread.h>
3  #include <unistd.h>
4  #define NUM_ITERATIONS 1000000
5  int count=10;
6  pthread_mutex_t mutex; // mutex object
7  // Critical section function
8  void critical_section(int process) {
9      //printf("Process %d is in the critical section\n", process);
10     //sleep(1); // Simulate some work in the critical section
11     if(process==0){
12
13         for (int i = 0; i < NUM_ITERATIONS; i++)
14             count--;
15     }
16     else
17     {
18         for (int i = 0; i < NUM_ITERATIONS; i++)
19             count++;
20     }
21     //printf("Process %d has updated count to %d\n", process, count);
22     //printf("Process %d is leaving the critical section\n", process);
23 }
24 // Peterson's Algorithm function for process 0
25 void *process0(void *arg) {
26     pthread_mutex_lock(&mutex); // lock
27     // Critical section
28     critical_section(0);
29     // Exit section
30     pthread_mutex_unlock(&mutex); // unlock
31     return NULL;
32 }
33 // Peterson's Algorithm function for process 1
34 void *process1(void *arg) {
35     pthread_mutex_lock(&mutex); // lock
36     // Critical section
37     critical_section(1);
38     // Exit section
39     pthread_mutex_unlock(&mutex); // unlock
40     return NULL;
41 }
42 void *process2(void *arg) {
43     pthread_mutex_lock(&mutex); // lock
44     // Critical section
45     critical_section(2);
46     // Exit section
47     pthread_mutex_unlock(&mutex); // unlock
48     return NULL;
49 }
50 int main() {
51     pthread_t thread0, thread1, thread2;
52     pthread_mutex_init(&mutex, NULL); // initialize mutex
53     // Create threads
54     pthread_create(&thread0, NULL, process0, NULL);
55     pthread_create(&thread1, NULL, process1, NULL);
56     pthread_create(&thread2, NULL, process2, NULL);
57     // Wait for threads to finish
58     pthread_join(thread0, NULL);
59     pthread_join(thread1, NULL);
60     pthread_join(thread2, NULL);
61     pthread_mutex_destroy(&mutex); // destroy mutex
62     printf("Final count: %d\n", count);
63     return 0;
64 }

```

Output:

The screenshot shows a Visual Studio Code editor window titled 'LAB-6-1148 [WSL: Ubuntu-24.04]'. The Explorer panel on the left shows a file tree for 'LAB-6-1148 [WSL: UBUNT...]' containing files Task1.c, Task1.out, Task2.c, Task2.out, Task3.c, Task3.out, Task4.c, Task4.out, Task5.c, and Task5.out. The main editor area displays the code for Task5.c:

```

1 #include <stdio.h>
3 #include <unistd.h>
4 #define NUM_ITERATIONS 1000000
5 int count=10;

```

Below the code, the TERMINAL panel shows the following commands and output:

```

ohail_arwar@SALMAN:~/OS-1148/LAB-6-1148$ gcc Task5.c -o Task5.out
ohail_arwar@SALMAN:~/OS-1148/LAB-6-1148$ ./Task5.out
Final count: 1000010
ohail_arwar@SALMAN:~/OS-1148/LAB-6-1148$

```

Peterson	Mutex
It is software-based synchronization	it is hardware-based synchronization.
Achieving mutual exclusion between two processes only.	Achieving mutual exclusion between two or more threads/processes.
It uses two shared variables i.e., flag[] and turn.	It uses system calls like lock() and unlock().
It supported only two processes.	Multiple threads and processes easily supported.
It works only on sequentially consistent memory systems.	It uses hardware level atomic operations.
Example: flag [i] = true; turn = j; While(flag [j] && turn == j);	Example: pthread_mutex_lock(&lock)... Pthread_mutex_unlock(&lock)...